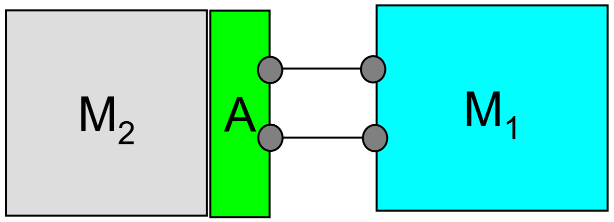
Generating Assumptions

Does system made up of modules M_1 and M_2 satisfy property P?

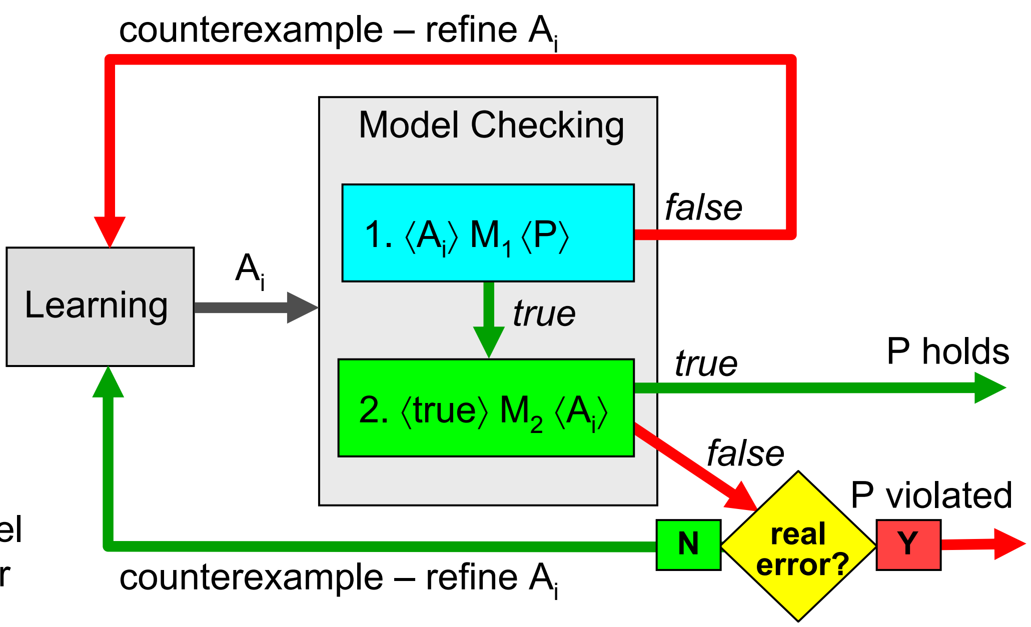


- Check P on entire system: too many states
- Check a module at a time: need abstractions of other modules
- Assume-guarantee approach:
- 1. check P on M_1 with *assumption* A for M_2
- 2. check that M_2 satisfies A

  } P holds in system
- Assumption generation is a manual process

Our work: the first *incremental* and *automated* approach for assume-guarantee reasoning

- Process is *iterative*
- We use a learning algorithm to infer the *smallest* assumption at each stage
- Assumptions are generated by querying the system, and are gradually refined as needed
- Queries are answered by model checking
- Refinement is based on counterexamples obtained by model checking
- Termination is guaranteed
- Algorithm is "any-time"
- Approach implemented in design-level model checker, and partly in software model checker



counterexample – refine A_i
Model Checking
Learning
A_i
1. $\langle A_i \rangle\ M_1\ \langle P \rangle$
false
true
2. $\langle true \rangle\ M_2\ \langle A_i \rangle$
true
P holds
false
P violated
N
real error?
Y
counterexample – refine A_i

Explanation

- **POC:** Dimitra Giannakopoulou and Corina Pasareanu
- **Background:** Verification of large systems requires a divide and conquer approach; system properties are checked in terms of properties of components. The correctness of a component depends on the behavior of components with which it interacts. Assume-guarantee reasoning is a divide and conquer approach to verification that makes use of “assumptions”, i.e., abstractions of the environment of a component. Although aimed at scalability, such reasoning has not been widely applied, because coming up with appropriate assumptions is a difficult manual process.
- **Accomplishment:** We have developed a novel framework for incremental, and fully automated assume-guarantee reasoning. The basis is the use of a learning algorithm to generate assumptions automatically. Our framework has been implemented in the LTSA tool for design-level model checking. It has been successfully applied to a number of case studies, including the Mars K9 Executive prototype.
- **Shown on Slide:** To check a property on a component, first check that if the component is part of an environment that satisfies some assumption, then the component satisfies the property. What remains to be shown is that the environment satisfies the assumption. We use a learning algorithm to generate assumptions automatically. Assumptions are initially approximate, but become gradually more precise by means of counterexamples obtained by model checking the component and its environment alternately. The algorithm is “any time”; it produces intermediate assumptions and is guaranteed to terminate with precise answers.
- **Future Plans:** Include implementing our framework in JPF, evaluating our approach in the context of large programs, improving the efficiency and memory usage of our algorithms, and extending the types of properties that our framework handles.